

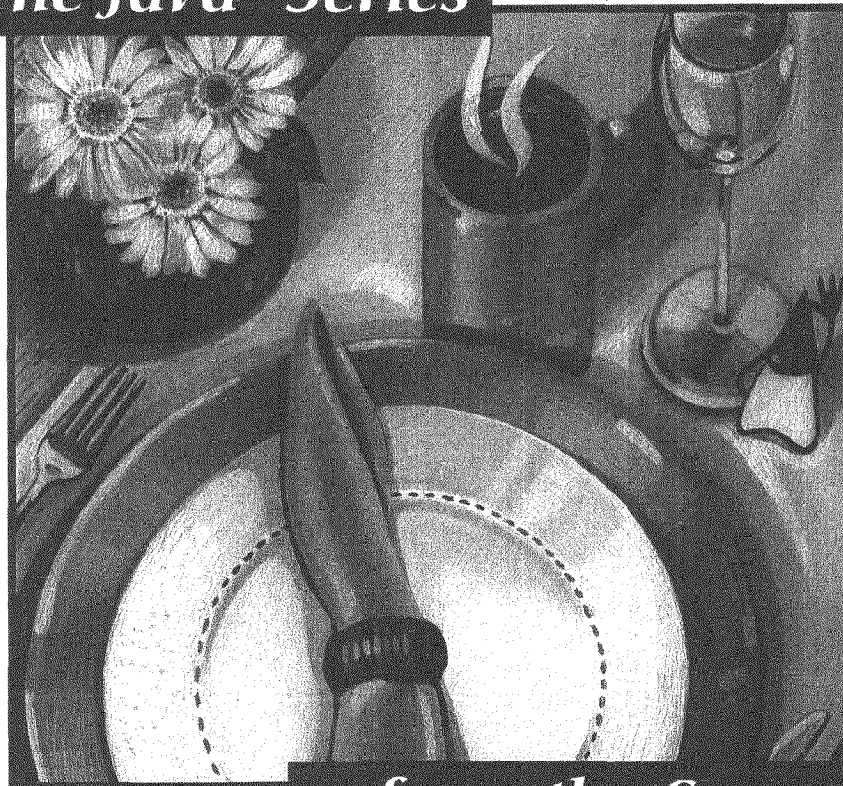
Exhibit C

James Gosling • Bill Joy • Guy Steele • Gilad Bracha



The Java™ Language Specification, Third Edition

The Java™ Series



...from the Source™



UNITED STATES DISTRICT COURT
NORTHERN DISTRICT OF CALIFORNIA

TRIAL EXHIBIT 984

CASE NO. 10-03561 WHA

DATE ENTERED _____

BY _____

DEPUTY CLERK



The Java™ Language Specification, Third Edition

Gosling
Joy
Steele
Bracha



The Java™
Language Specification
Third Edition

The Java™ Language Specification *Third Edition*

James Gosling
Bill Joy
Guy Steele
Gilad Bracha

◆Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Copyright © 1996-2005 Sun Microsystems, Inc.
4150 Network Circle, Santa Clara, California 95054 U.S.A.
All rights reserved.

Duke logo™ designed by Joe Palrang.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the United States Government is subject to the restrictions set forth in DFARS 252.227-7013 (c)(1)(ii) and FAR 52.227-19.

The release described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

Sun Microsystems, Inc. (SUN) hereby grants to you a fully paid, nonexclusive, nontransferable, perpetual, worldwide limited license (without the right to sublicense) under SUN's intellectual property rights that are essential to practice this specification. This license allows and is limited to the creation and distribution of clean room implementations of this specification that: (i) include a complete implementation of the current version of this specification without subsetting or supersetting; (ii) implement all the interfaces and functionality of the required packages of the Java™ 2 Platform, Standard Edition, as defined by SUN, without subsetting or supersetting; (iii) do not add any additional packages, classes, or interfaces to the java.* or javax.* packages or their subpackages; (iv) pass all test suites relating to the most recent published version of the specification of the Java™ 2 Platform, Standard Edition, that are available from SUN six (6) months prior to any beta release of the clean room implementation or upgrade thereto; (v) do not derive from SUN source code or binary materials; and (vi) do not include any SUN source code or binary materials without an appropriate and separate license from SUN.

Sun, Sun Microsystems, the Sun logo, Solaris, Java, JavaScript, JDK, and all Java-based trademarks or logos are trademarks or registered trademarks of Sun Microsystems, Inc. UNIX® is a registered trademark of The Open Group in the United States and other countries. Apple and Dylan are trademarks of Apple Computer, Inc. All other product names mentioned herein are the trademarks of their respective owners.

THIS PUBLICATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. SUN MICROSYSTEMS, INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS PUBLICATION AT ANY TIME.

Credits and permissions for quoted material appear in a separate section on page 649.

ISBN 0-321-24678-0

This product is printed digitally on demand.

First printing, May 2005

“When *I* use a word,” Humpty Dumpty said, in rather a scornful tone, “it means just what I choose it to mean—neither more nor less.”

“The question is,” said Alice, “whether you *can* make words mean so many different things.”

“The question is,” said Humpty Dumpty, “which is to be master—that’s all.”

—Lewis Carroll, *Through the Looking Glass*

Table of Contents

Preface xxi

Preface to the Second Edition xxv

Preface to the Third Edition xxix

1 Introduction 1

- 1.1 Example Programs 5
- 1.2 Notation 6
- 1.3 Relationship to Predefined Classes and Interfaces 6
- 1.4 References 6

2 Grammars 9

- 2.1 Context-Free Grammars 9
- 2.2 The Lexical Grammar 9
- 2.3 The Syntactic Grammar 10
- 2.4 Grammar Notation 10

3 Lexical Structure 13

- 3.1 Unicode 13
- 3.2 Lexical Translations 14
- 3.3 Unicode Escapes 15
- 3.4 Line Terminators 16
- 3.5 Input Elements and Tokens 17
- 3.6 White Space 18
- 3.7 Comments 18
- 3.8 Identifiers 19
- 3.9 Keywords 21
- 3.10 Literals 21
 - 3.10.1 Integer Literals 22
 - 3.10.2 Floating-Point Literals 24
 - 3.10.3 Boolean Literals 26

TABLE OF CONTENTS

3.10.4	Character Literals	26
3.10.5	String Literals	28
3.10.6	Escape Sequences for Character and String Literals	30
3.10.7	The Null Literal	30
3.11	Separators	31
3.12	Operators	31
4	Types, Values, and Variables	33
4.1	The Kinds of Types and Values	34
4.2	Primitive Types and Values	34
4.2.1	Integral Types and Values	35
4.2.2	Integer Operations	36
4.2.3	Floating-Point Types, Formats, and Values	37
4.2.4	Floating-Point Operations	40
4.2.5	The boolean Type and boolean Values	43
4.3	Reference Types and Values	44
4.3.1	Objects	45
4.3.2	The Class Object	47
4.3.3	The Class String	48
4.3.4	When Reference Types Are the Same	49
4.4	Type Variables	49
4.5	Parameterized Types	51
4.5.1	Type Arguments and Wildcards	52
4.5.1.1	Type Argument Containment and Equivalence	55
4.5.2	Members and Constructors of Parameterized Types	55
4.6	Type Erasure	56
4.7	Reifiable Types	56
4.8	Raw Types	57
4.9	Intersection Types	62
4.10	Subtyping	63
4.10.1	Subtyping among Primitive Types	63
4.10.2	Subtyping among Class and Interface Types	63
4.10.3	Subtyping among Array Types	64
4.11	Where Types Are Used	65
4.12	Variables	67
4.12.1	Variables of Primitive Type	67
4.12.2	Variables of Reference Type	67
4.12.2.1	Heap Pollution	68
4.12.3	Kinds of Variables	69
4.12.4	final Variables	71
4.12.5	Initial Values of Variables	71
4.12.6	Types, Classes, and Interfaces	73
5	Conversions and Promotions	77
5.1	Kinds of Conversion	80
5.1.1	Identity Conversions	80
5.1.2	Widening Primitive Conversion	80

TABLE OF CONTENTS

5.1.3	Narrowing Primitive Conversions	82
5.1.4	Widening and Narrowing Primitive Conversions	84
5.1.5	Widening Reference Conversions	85
5.1.6	Narrowing Reference Conversions	85
5.1.7	Boxing Conversion	86
5.1.8	Unboxing Conversion	88
5.1.9	Unchecked Conversion	89
5.1.10	Capture Conversion	89
5.1.11	String Conversions	92
5.1.12	Forbidden Conversions	92
5.1.13	Value Set Conversion	92
5.2	Assignment Conversion	93
5.3	Method Invocation Conversion	99
5.4	String Conversion	101
5.5	Casting Conversion	101
5.6	Numeric Promotions	108
5.6.1	Unary Numeric Promotion	108
5.6.2	Binary Numeric Promotion	110
6	Names	113
6.1	Declarations	114
6.2	Names and Identifiers	115
6.3	Scope of a Declaration	117
6.3.1	Shadowing Declarations	119
6.3.2	Obscured Declarations	122
6.4	Members and Inheritance	122
6.4.1	The Members of Type Variables, Parameterized Types, Raw Types and Intersection Types	122
6.4.2	The Members of a Package	122
6.4.3	The Members of a Class Type	123
6.4.4	The Members of an Interface Type	124
6.4.5	The Members of an Array Type	125
6.5	Determining the Meaning of a Name	126
6.5.1	Syntactic Classification of a Name According to Context	127
6.5.2	Reclassification of Contextually Ambiguous Names	129
6.5.3	Meaning of Package Names	131
6.5.3.1	Simple Package Names	131
6.5.3.2	Qualified Package Names	132
6.5.4	Meaning of <i>PackageOrTypeNames</i>	132
6.5.4.1	Simple <i>PackageOrTypeNames</i>	132
6.5.4.2	Qualified <i>PackageOrTypeNames</i>	132
6.5.5	Meaning of Type Names	132
6.5.5.1	Simple Type Names	132
6.5.5.2	Qualified Type Names	132
6.5.6	Meaning of Expression Names	134
6.5.6.1	Simple Expression Names	134
6.5.6.2	Qualified Expression Names	135
6.5.7	Meaning of Method Names	137

TABLE OF CONTENTS

	6.5.7.1	Simple Method Names	137
	6.5.7.2	Qualified Method Names	137
6.6	Access Control		138
	6.6.1	Determining Accessibility	138
	6.6.2	Details on protected Access	139
	6.6.2.1	Access to a protected Member	139
	6.6.2.2	Qualified Access to a protected Constructor	140
	6.6.3	An Example of Access Control	140
	6.6.4	Example: Access to public and Non-public Classes	141
	6.6.5	Example: Default-Access Fields, Methods, and Constructors	142
	6.6.6	Example: public Fields, Methods, and Constructors	143
	6.6.7	Example: protected Fields, Methods, and Constructors	143
	6.6.8	Example: private Fields, Methods, and Constructors	144
6.7	Fully Qualified Names and Canonical Names		145
6.8	Naming Conventions		146
	6.8.1	Package Names	147
	6.8.2	Class and Interface Type Names	147
	6.8.3	Type Variable Names	148
	6.8.4	Method Names	149
	6.8.5	Field Names	150
	6.8.6	Constant Names	150
	6.8.7	Local Variable and Parameter Names	151
7	Packages		153
	7.1	Package Members	154
	7.2	Host Support for Packages	155
	7.2.1	Storing Packages in a File System	155
	7.2.2	Storing Packages in a Database	157
	7.3	Compilation Units	157
	7.4	Package Declarations	158
	7.4.1	Named Packages	158
	7.4.1.1	Package Annotations	158
	7.4.2	Unnamed Packages	159
	7.4.3	Observability of a Package	160
	7.4.4	Scope of a Package Declaration	160
	7.5	Import Declarations	160
	7.5.1	Single-Type-Import Declaration	161
	7.5.2	Type-Import-on-Demand Declaration	163
	7.5.3	Single Static Import Declaration	164
	7.5.4	Static-Import-on-Demand Declaration	165
	7.5.5	Automatic Imports	165
	7.5.6	A Strange Example	165
	7.6	Top Level Type Declarations	166
	7.7	Unique Package Names	169
8	Classes		173
	8.1	Class Declaration	175

TABLE OF CONTENTS

8.1.1	Class Modifiers	175
8.1.1.1	abstract Classes	176
8.1.1.2	final Classes	178
8.1.1.3	strictfp Classes	178
8.1.2	Generic Classes and Type Parameters	178
8.1.3	Inner Classes and Enclosing Instances	181
8.1.4	Superclasses and Subclasses	184
8.1.5	Superinterfaces	186
8.1.6	Class Body and Member Declarations	189
8.2	Class Members	190
8.2.1	Examples of Inheritance	192
8.2.1.1	Example: Inheritance with Default Access	192
8.2.1.2	Inheritance with public and protected	193
8.2.1.3	Inheritance with private	193
8.2.1.4	Accessing Members of Inaccessible Classes	194
8.3	Field Declarations	196
8.3.1	Field Modifiers	197
8.3.1.1	static Fields	198
8.3.1.2	final Fields	199
8.3.1.3	transient Fields	199
8.3.1.4	volatile Fields	199
8.3.2	Initialization of Fields	201
8.3.2.1	Initializers for Class Variables	202
8.3.2.2	Initializers for Instance Variables	202
8.3.2.3	Restrictions on the use of Fields during Initialization	203
8.3.3	Examples of Field Declarations	205
8.3.3.1	Example: Hiding of Class Variables	205
8.3.3.2	Example: Hiding of Instance Variables	206
8.3.3.3	Example: Multiply Inherited Fields	207
8.3.3.4	Example: Re-inheritance of Fields	209
8.4	Method Declarations	209
8.4.1	Formal Parameters	210
8.4.2	Method Signature	212
8.4.3	Method Modifiers	214
8.4.3.1	abstract Methods	214
8.4.3.2	static Methods	216
8.4.3.3	final Methods	217
8.4.3.4	native Methods	218
8.4.3.5	strictfp Methods	218
8.4.3.6	synchronized Methods	218
8.4.4	Generic Methods	220
8.4.5	Method Return Type	220
8.4.6	Method Throws	221
8.4.7	Method Body	223
8.4.8	Inheritance, Overriding, and Hiding	224
8.4.8.1	Overriding (by Instance Methods)	224
8.4.8.2	Hiding (by Class Methods)	225
8.4.8.3	Requirements in Overriding and Hiding	225

TABLE OF CONTENTS

	8.4.8.4	Inheriting Methods with Override-Equivalent Signatures	228
	8.4.9	Overloading	229
	8.4.10	Examples of Method Declarations	230
		8.4.10.1 Example: Overriding	230
		8.4.10.2 Example: Overloading, Overriding, and Hiding	231
		8.4.10.3 Example: Incorrect Overriding	231
		8.4.10.4 Example: Overriding versus Hiding	232
		8.4.10.5 Example: Invocation of Hidden Class Methods	234
		8.4.10.6 Large Example of Overriding	234
		8.4.10.7 Example: Incorrect Overriding because of Throws	236
8.5		Member Type Declarations	237
	8.5.1	Modifiers	238
	8.5.2	Static Member Type Declarations	238
8.6		Instance Initializers	238
8.7		Static Initializers	239
8.8		Constructor Declarations	240
	8.8.1	Formal Parameters and Formal Type Parameter	240
	8.8.2	Constructor Signature	241
	8.8.3	Constructor Modifiers	241
	8.8.4	Generic Constructors	242
	8.8.5	Constructor Throws	242
	8.8.6	The Type of a Constructor	242
	8.8.7	Constructor Body	242
		8.8.7.1 Explicit Constructor Invocations	243
	8.8.8	Constructor Overloading	246
	8.8.9	Default Constructor	247
	8.8.10	Preventing Instantiation of a Class	248
8.9		Enums	249
9		Interfaces	259
	9.1	Interface Declarations	260 -
		9.1.1 Interface Modifiers	260
			9.1.1.1 abstract Interfaces 261
			9.1.1.2 strictfp Interfaces 261
		9.1.2 Generic Interfaces and Type Parameters	261
		9.1.3 Superinterfaces and Subinterfaces	261
		9.1.4 Interface Body and Member Declarations	263
		9.1.5 Access to Interface Member Names	263
	9.2	Interface Members	263
	9.3	Field (Constant) Declarations	264
		9.3.1 Initialization of Fields in Interfaces	265
		9.3.2 Examples of Field Declarations	265
			9.3.2.1 Ambiguous Inherited Fields 265
			9.3.2.2 Multiply Inherited Fields 266
	9.4	Abstract Method Declarations	266
		9.4.1 Inheritance and Overriding	267
		9.4.2 Overloading	268

TABLE OF CONTENTS

9.4.3	Examples of Abstract Method Declarations	269
9.4.3.1	Example: Overriding	269
9.4.3.2	Example: Overloading	269
9.5	Member Type Declarations	270
9.6	Annotation Types	270
9.6.1	Predefined Annotation Types	277
9.6.1.1	Target	278
9.6.1.2	Retention	278
9.6.1.3	Inherited	279
9.6.1.4	Override	279
9.6.1.5	SuppressWarnings	280
9.6.1.6	Deprecated	280
9.7	Annotations	281
10	Arrays	287
10.1	Array Types	288
10.2	Array Variables	288
10.3	Array Creation	289
10.4	Array Access	289
10.5	Arrays: A Simple Example	290
10.6	Array Initializers	290
10.7	Array Members	292
10.8	Class Objects for Arrays	293
10.9	An Array of Characters is Not a String	294
10.10	Array Store Exception	294
11	Exceptions	297
11.1	The Causes of Exceptions	298
11.2	Compile-Time Checking of Exceptions	299
11.2.1	Exception Analysis of Expressions	299
11.2.2	Exception Analysis of Statements	300
11.2.3	Exception Checking	301
11.2.4	Why Errors are Not Checked	301
11.2.5	Why Runtime Exceptions are Not Checked	301
11.3	Handling of an Exception	302
11.3.1	Exceptions are Precise	303
11.3.2	Handling Asynchronous Exceptions	303
11.4	An Example of Exceptions	304
11.5	The Exception Hierarchy	306
11.5.1	Loading and Linkage Errors	307
11.5.2	Virtual Machine Errors	307
12	Execution	309
12.1	Virtual Machine Start-Up	309
12.1.1	Load the Class Test	310
12.1.2	Link Test: Verify, Prepare, (Optionally) Resolve	310

TABLE OF CONTENTS

12.1.3	Initialize Test: Execute Initializers	311
12.1.4	Invoke Test.main	312
12.2	Loading of Classes and Interfaces	312
12.2.1	The Loading Process	313
12.3	Linking of Classes and Interfaces	314
12.3.1	Verification of the Binary Representation	314
12.3.2	Preparation of a Class or Interface Type	315
12.3.3	Resolution of Symbolic References	315
12.4	Initialization of Classes and Interfaces	316
12.4.1	When Initialization Occurs	316
12.4.2	Detailed Initialization Procedure	319
12.4.3	Initialization: Implications for Code Generation	321
12.5	Creation of New Class Instances	322
12.6	Finalization of Class Instances	325
12.6.1	Implementing Finalization	326
12.6.1.1	Interaction with the Memory Model	328
12.6.2	Finalizer Invocations are Not Ordered	329
12.7	Unloading of Classes and Interfaces	330
12.8	Program Exit	331

13 Binary Compatibility 333

13.1	The Form of a Binary	334
13.2	What Binary Compatibility Is and Is Not	339
13.3	Evolution of Packages	340
13.4	Evolution of Classes	340
13.4.1	abstract Classes	340
13.4.2	final Classes	341
13.4.3	public Classes	341
13.4.4	Superclasses and Superinterfaces	341
13.4.5	Class Formal Type Parameters	342
13.4.6	Class Body and Member Declarations	343
13.4.7	Access to Members and Constructors	344
13.4.8	Field Declarations	345
13.4.9	final Fields and Constants	347
13.4.10	static Fields	349
13.4.11	transient Fields	350
13.4.12	Method and Constructor Declarations	350
13.4.13	Method and Constructor Formal Type Parameters	351
13.4.14	Method and Constructor Parameters	352
13.4.15	Method Result Type	352
13.4.16	abstract Methods	352
13.4.17	final Methods	353
13.4.18	native Methods	354
13.4.19	static Methods	354
13.4.20	synchronized Methods	354
13.4.21	Method and Constructor Throws	354
13.4.22	Method and Constructor Body	354
13.4.23	Method and Constructor Overloading	355

TABLE OF CONTENTS

13.4.24	Method Overriding	356
13.4.25	Static Initializers	356
13.4.26	Evolution of Enums	356
13.5	Evolution of Interfaces	356
13.5.1	public Interfaces	356
13.5.2	Superinterfaces	357
13.5.3	The Interface Members	357
13.5.4	Interface Formal Type Parameters	357
13.5.5	Field Declarations	358
13.5.6	Abstract Method Declarations	358
13.5.7	Evolution of Annotation Types	358

14 Blocks and Statements 359

14.1	Normal and Abrupt Completion of Statements	360
14.2	Blocks	361
14.3	Local Class Declarations	361
14.4	Local Variable Declaration Statements	363
14.4.1	Local Variable Declarators and Types	364
14.4.2	Scope of Local Variable Declarations	364
14.4.3	Shadowing of Names by Local Variables	367
14.4.4	Execution of Local Variable Declarations	367
14.5	Statements	368
14.6	The Empty Statement	370
14.7	Labeled Statements	370
14.8	Expression Statements	371
14.9	The if Statement	372
14.9.1	The if-then Statement	372
14.9.2	The if-then-else Statement	372
14.10	The assert Statement	373
14.11	The switch Statement	377
14.12	The while Statement	380
14.12.1	Abrupt Completion	381
14.13	The do Statement	382
14.13.1	Abrupt Completion	383
14.13.2	Example of do statement	383
14.14	The for Statement	384
14.14.1	The basic for Statement	384
14.14.1.1	Initialization of for statement	385
14.14.1.2	Iteration of for statement	385
14.14.1.3	Abrupt Completion of for statement	386
14.14.2	The enhanced for statement	387
14.15	The break Statement	388
14.16	The continue Statement	390
14.17	The return Statement	392
14.18	The throw Statement	393
14.19	The synchronized Statement	395
14.20	The try statement	396
14.20.1	Execution of try-catch	398

TABLE OF CONTENTS

- 14.20.2 Execution of try-catch-finally 399
- 14.21 Unreachable Statements 402

15 Expressions 409

- 15.1 Evaluation, Denotation, and Result 409
- 15.2 Variables as Values 410
- 15.3 Type of an Expression 410
- 15.4 FP-strict Expressions 411
- 15.5 Expressions and Run-Time Checks 411
- 15.6 Normal and Abrupt Completion of Evaluation 413
- 15.7 Evaluation Order 414
 - 15.7.1 Evaluate Left-Hand Operand First 415
 - 15.7.2 Evaluate Operands before Operation 416
 - 15.7.3 Evaluation Respects Parentheses and Precedence 417
 - 15.7.4 Argument Lists are Evaluated Left-to-Right 418
 - 15.7.5 Evaluation Order for Other Expressions 419
- 15.8 Primary Expressions 420
 - 15.8.1 Lexical Literals 420
 - 15.8.2 Class Literals 421
 - 15.8.3 `this` 421
 - 15.8.4 Qualified `this` 422
 - 15.8.5 Parenthesized Expressions 422
- 15.9 Class Instance Creation Expressions 423
 - 15.9.1 Determining the Class being Instantiated 424
 - 15.9.2 Determining Enclosing Instances 425
 - 15.9.3 Choosing the Constructor and its Arguments 427
 - 15.9.4 Run-time Evaluation of Class Instance Creation Expressions 428
 - 15.9.5 Anonymous Class Declarations 429
 - 15.9.5.1 Anonymous Constructors 429
 - 15.9.6 Example: Evaluation Order and Out-of-Memory Detection 430
- 15.10 Array Creation Expressions 431
 - 15.10.1 Run-time Evaluation of Array Creation Expressions 432
 - 15.10.2 Example: Array Creation Evaluation Order 433
 - 15.10.3 Example: Array Creation and Out-of-Memory Detection 434
- 15.11 Field Access Expressions 435
 - 15.11.1 Field Access Using a Primary 435
 - 15.11.2 Accessing Superclass Members using `super` 438
- 15.12 Method Invocation Expressions 440
 - 15.12.1 Compile-Time Step 1: Determine Class or Interface to Search 440
 - 15.12.2 Compile-Time Step 2: Determine Method Signature 442
 - 15.12.2.1 Identify Potentially Applicable Methods 443
 - 15.12.2.2 Phase 1: Identify Matching Arity Methods Applicable by Subtyping 445
 - 15.12.2.3 Phase 2: Identify Matching Arity Methods Applicable by Method Invocation Conversion 446
 - 15.12.2.4 Phase 3: Identify Applicable Variable Arity Methods 446
 - 15.12.2.5 Choosing the Most Specific Method 447
 - 15.12.2.6 Method Result and Throws Types 450

TABLE OF CONTENTS

15.12.2.7	Inferring Type Arguments Based on Actual Arguments	451
15.12.2.8	Inferring Unresolved Type Arguments	466
15.12.2.9	Examples	466
15.12.2.10	Example: Overloading Ambiguity	468
15.12.2.11	Example: Return Type Not Considered	468
15.12.2.12	Example: Compile-Time Resolution	469
15.12.3	Compile-Time Step 3: Is the Chosen Method Appropriate?	471
15.12.4	Runtime Evaluation of Method Invocation	473
15.12.4.1	Compute Target Reference (If Necessary)	473
15.12.4.2	Evaluate Arguments	474
15.12.4.3	Check Accessibility of Type and Method	475
15.12.4.4	Locate Method to Invoke	476
15.12.4.5	Create Frame, Synchronize, Transfer Control	477
15.12.4.6	Example: Target Reference and Static Methods	479
15.12.4.7	Example: Evaluation Order	479
15.12.4.8	Example: Overriding	480
15.12.4.9	Example: Method Invocation using super	481
15.13	Array Access Expressions	482
15.13.1	Runtime Evaluation of Array Access	483
15.13.2	Examples: Array Access Evaluation Order	483
15.14	Postfix Expressions	485
15.14.1	Expression Names	485
15.14.2	Postfix Increment Operator ++	485
15.14.3	Postfix Decrement Operator --	486
15.15	Unary Operators	487
15.15.1	Prefix Increment Operator ++	487
15.15.2	Prefix Decrement Operator --	488
15.15.3	Unary Plus Operator +	489
15.15.4	Unary Minus Operator -	489
15.15.5	Bitwise Complement Operator ~	490
15.15.6	Logical Complement Operator !	490
15.16	Cast Expressions	490
15.17	Multiplicative Operators	491
15.17.1	Multiplication Operator *	492
15.17.2	Division Operator /	493
15.17.3	Remainder Operator %	495
15.18	Additive Operators	496
15.18.1	String Concatenation Operator +	497
15.18.1.1	String Conversion	497
15.18.1.2	Optimization of String Concatenation	498
15.18.1.3	Examples of String Concatenation	498
15.18.2	Additive Operators (+ and -) for Numeric Types	500
15.19	Shift Operators	502
15.20	Relational Operators	503
15.20.1	Numerical Comparison Operators <, <=, >, and >=	503
15.20.2	Type Comparison Operator instanceof	504
15.21	Equality Operators	505

TABLE OF CONTENTS

15.21.1	Numerical Equality Operators == and !=	506
15.21.2	Boolean Equality Operators == and !=	507
15.21.3	Reference Equality Operators == and !=	507
15.22	Bitwise and Logical Operators	508
15.22.1	Integer Bitwise Operators &, ^, and	508
15.22.2	Boolean Logical Operators &, ^, and	508
15.23	Conditional-And Operator &&	509
15.24	Conditional-Or Operator	509
15.25	Conditional Operator ?	510
15.26	Assignment Operators	512
15.26.1	Simple Assignment Operator =	513
15.26.2	Compound Assignment Operators	518
15.27	Expression	525
15.28	Constant Expression	525

16 Definite Assignment 527

16.1	Definite Assignment and Expressions	533
16.1.1	Boolean Constant Expressions	533
16.1.2	The Boolean Operator &&	533
16.1.3	The Boolean Operator	534
16.1.4	The Boolean Operator !	534
16.1.5	The Boolean Operator ?	534
16.1.6	The Conditional Operator ?	535
16.1.7	Other Expressions of Type boolean	535
16.1.8	Assignment Expressions	535
16.1.9	Operators ++ and --	536
16.1.10	Other Expressions	536
16.2	Definite Assignment and Statements	538
16.2.1	Empty Statements	538
16.2.2	Blocks	538
16.2.3	Local Class Declaration Statements	539
16.2.4	Local Variable Declaration Statements	539
16.2.5	Labeled Statements	540
16.2.6	Expression Statements	540
16.2.7	if Statements	541
16.2.8	assert Statements	541
16.2.9	switch Statements	541
16.2.10	while Statements	542
16.2.11	do Statements	543
16.2.12	for Statements	543
16.2.12.1	Initialization Part	544
16.2.12.2	Incrementation Part	544
16.2.13	break, continue, return, and throw Statements	545
16.2.14	synchronized Statements	545
16.2.15	try Statements	545
16.3	Definite Assignment and Parameters	547
16.4	Definite Assignment and Array Initializers	547
16.5	Definite Assignment and Enum Constants	548

TABLE OF CONTENTS

- 16.6 Definite Assignment and Anonymous Classes 548
- 16.7 Definite Assignment and Member Types 549
- 16.8 Definite Assignment and Static Initializers 549
- 16.9 Definite Assignment, Constructors, and Instance Initializers 550

17 Threads and Locks 553

- 17.1 Locks 554
- 17.2 Notation in Examples 554
- 17.3 Incorrectly Synchronized Programs Exhibit Surprising Behaviors 555
- 17.4 Memory Model 557
 - 17.4.1 Shared Variables 558
 - 17.4.2 Actions 558
 - 17.4.3 Programs and Program Order 560
 - 17.4.4 Synchronization Order 561
 - 17.4.5 Happens-before Order 561
 - 17.4.6 Executions 567
 - 17.4.7 Well-Formed Executions 568
 - 17.4.8 Executions and Causality Requirements 568
 - 17.4.9 Observable Behavior and Nonterminating Executions 571
- 17.5 Final Field Semantics 573
 - 17.5.1 Semantics of Final Fields 575
 - 17.5.2 Reading Final Fields During Construction 576
 - 17.5.3 Subsequent Modification of Final Fields 576
 - 17.5.4 Write Protected Fields 578
- 17.6 Word Tearing 578
- 17.7 Non-atomic Treatment of double and long 579
- 17.8 Wait Sets and Notification 580
 - 17.8.1 Wait 580
 - 17.8.2 Notification 581
 - 17.8.3 Interruptions 582
 - 17.8.4 Interactions of Waits, Notification and Interruption 582
- 17.9 Sleep and Yield 583

18 Syntax 585

- 18.1 The Grammar of the Java Programming Language 585

Index 597

Credits 649

Colophon 651

Preface

THE Java™ programming language was originally called Oak, and was designed for use in embedded consumer-electronic applications by James Gosling. After several years of experience with the language, and significant contributions by Ed Frank, Patrick Naughton, Jonathan Payne, and Chris Warth it was retargeted to the Internet, renamed, and substantially revised to be the language specified here. The final form of the language was defined by James Gosling, Bill Joy, Guy Steele, Richard Tuck, Frank Yellin, and Arthur van Hoff, with help from Graham Hamilton, Tim Lindholm, and many other friends and colleagues.

The Java programming language is a general-purpose concurrent class-based object-oriented programming language, specifically designed to have as few implementation dependencies as possible. It allows application developers to write a program once and then be able to run it everywhere on the Internet.

This book attempts a complete specification of the syntax and semantics of the language. We intend that the behavior of every language construct is specified here, so that all implementations will accept the same programs. Except for timing dependencies or other non-determinisms and given sufficient time and sufficient memory space, a program written in the Java programming language should compute the same result on all machines and in all implementations.

We believe that the Java programming language is a mature language, ready for widespread use. Nevertheless, we expect some evolution of the language in the years to come. We intend to manage this evolution in a way that is completely compatible with existing applications. To do this, we intend to make relatively few new versions of the language. Compilers and systems will be able to support the several versions simultaneously, with complete compatibility.

Much research and experimentation with the Java platform is already underway. We encourage this work, and will continue to cooperate with external groups to explore improvements to the language and platform. For example, we have already received several interesting proposals for parameterized types. In technically difficult areas, near the state of the art, this kind of research collaboration is essential.

PREFACE

We acknowledge and thank the many people who have contributed to this book through their excellent feedback, assistance and encouragement:

Particularly thorough, careful, and thoughtful reviews of drafts were provided by Tom Cargill, Peter Deutsch, Paul Hilfinger, Masayuki Ida, David Moon, Steven Muchnick, Charles L. Perkins, Chris Van Wyk, Steve Vinoski, Philip Wadler, Daniel Weinreb, and Kenneth Zadeck. We are very grateful for their extraordinary volunteer efforts.

We are also grateful for reviews, questions, comments, and suggestions from Stephen Adams, Bowen Alpern, Glenn Ammons, Leonid Arbuzov, Kim Bruce, Edwin Chan, David Chase, Pavel Curtis, Drew Dean, William Dietz, David Dill, Patrick Dussud, Ed Felten, John Giannandrea, John Gilmore, Charles Gust, Warren Harris, Lee Hasiuk, Mike Hendrickson, Mark Hill, Urs Hoelzle, Roger Hoover, Susan Flynn Hummel, Christopher Jang, Mick Jordan, Mukesh Kacker, Peter Kessler, James Larus, Derek Lieber, Bill McKeeman, Steve Naroff, Evi Nemeth, Robert O'Callahan, Dave Papay, Craig Partridge, Scott Pfeffer, Eric Raymond, Jim Roskind, Jim Russell, William Scherlis, Edith Schonberg, Anthony Scian, Matthew Self, Janice Shepherd, Kathy Stark, Barbara Steele, Rob Strom, William Waite, Greg Weeks, and Bob Wilson. (This list was generated semi-automatically from our E-mail records. We apologize if we have omitted anyone.)

The feedback from all these reviewers was invaluable to us in improving the definition of the language as well as the form of the presentation in this book. We thank them for their diligence. Any remaining errors in this book—we hope they are few—are our responsibility and not theirs.

We thank Francesca Freedman and Doug Kramer for assistance with matters of typography and layout. We thank Dan Mills of Adobe Systems Incorporated for assistance in exploring possible choices of typefaces.

Many of our colleagues at Sun Microsystems have helped us in one way or another. Lisa Friendly, our series editor, managed our relationship with Addison-Wesley. Susan Stambaugh managed the distribution of many hundreds of copies of drafts to reviewers. We received valuable assistance and technical advice from Ben Adida, Ole Agesen, Ken Arnold, Rick Cattell, Asmus Freytag, Norm Hardy, Steve Heller, David Hough, Doug Kramer, Nancy Lee, Marianne Mueller, Akira Tanaka, Greg Tarsy, David Ungar, Jim Waldo, Ann Wollrath, Geoff Wyant, and Derek White. We thank Alan Baratz, David Bowen, Mike Clary, John Doerr, Jon Kannegaard, Eric Schmidt, Bob Sproull, Bert Sutherland, and Scott McNealy for leadership and encouragement.

The on-line Bartleby Library of Columbia University, at URL:

<http://www.cc.columbia.edu/acis/bartleby/>

PREFACE

was invaluable to us during the process of researching and verifying many of the quotations that are scattered throughout this book. Here is one example:

They lard their lean books with the fat of others' works.

—Robert Burton (1576–1640)

We are grateful to those who have toiled on Project Bartleby, for saving us a great deal of effort and reawakening our appreciation for the works of Walt Whitman.

We are thankful for the tools and services we had at our disposal in writing this book: telephones, overnight delivery, desktop workstations, laser printers, photocopiers, text formatting and page layout software, fonts, electronic mail, the World Wide Web, and, of course, the Internet. We live in three different states, scattered across a continent, but collaboration with each other and with our reviewers has seemed almost effortless. Kudos to the thousands of people who have worked over the years to make these excellent tools and services work quickly and reliably.

Mike Hendrickson, Katie Duffy, Simone Payment, and Rosa Aimée González of Addison-Wesley were very helpful, encouraging, and patient during the long process of bringing this book to print. We also thank the copy editors.

Rosemary Simpson worked hard, on a very tight schedule, to create the index. We got into the act at the last minute, however; blame us and not her for any jokes you may find hidden therein.

Finally, we are grateful to our families and friends for their love and support during this last, crazy, year.

In their book *The C Programming Language*, Brian Kernighan and Dennis Ritchie said that they felt that the C language “wears well as one’s experience with it grows.” If you like C, we think you will like the Java programming language. We hope that it, too, wears well for you.

James Gosling
Cupertino, California

Bill Joy
Aspen, Colorado

Guy Steele
Chelmsford, Massachusetts

July, 1996